Webes adatkezelő környezetek

Sportegyesület

Takács Ákos
Neptun-kód: VTJ4ES
Miskolci Egyetem

2025

Tartalomjegyzék

1. A feladat leírása

A jelen jegyzőkönyv témája egy sportegyesület adatnyilvántartó rendszerének megtervezése és XML alapú megvalósítása. A feladat célja, hogy a sportklub működése során keletkező adatokat – sportolók adatai, edzők, edzéscsoportok, tagságok, helyszínek és versenyek – egységes, strukturált, jól validálható formában tároljuk. A projekt első részében a rendszer logikai modelljének megalkotása történik meg ER (Entity–Relationship) diagram segítségével. Ez a modell tartalmazza az egyedeket, azok tulajdonságait, valamint az egyedek közötti kapcsolatokat.

A második lépésben az ER modell XML-alapú XDM modellre történő konvertálása következik, amely megmutatja, hogy az adatmodell hogyan fordítható le XML elemekre, attribútumokra és struktúrákra. Ezután elkészül a sportegyesületet leíró konkrét XML dokumentum, amely több példányt is tartalmaz az ismétlődő elemekből (pl. sportolók, csoportok, versenyek). A dokumentumhoz XML séma (XSD) is tartozik, amelyben definiálva vannak a saját típusok, a kulcsok, az idegen kulcsok, valamint a komplex típusok.

A második fő feladat a DOM-alapú XML feldolgozás Java nyelven. Ez magában foglalja az XML dokumentum beolvasását, módosítását, lekérdezések végrehajtását, valamint a dokumentum kiíratását. A jegyzőkönyv részletezi a programozási lépéseket, kiemeli a fontos kódrészleteket, és ismerteti a DOM működésének főbb elemeit.

1.1. Az adatbázis ER modellje

A sportegyesülethez tartozó adatok logikai szerkezetét az ER (Entity–Relationship) modell írja le. A modellben összesen hat fő egyed található, amelyek a rendszer működéséhez szükséges különböző információcsoportokat reprezentálják: *Athlete*, *Coach*, *TrainingGroup*, *Location*, *Membership* és *Competition*. Az egyedekhez különböző attribútumok, többértékű tulajdonságok, valamint összetett tulajdonságok tartoznak.

Athlete (sportoló) egyed

A sportolók adatait tárolja. A név összetett tulajdonság (*first_name*, *last_name*), az *email* és *telefon* adatok pedig az elérhetőségek részét képezik. Továbbá a sportolókhoz többértékű *skills* tulajdonság tartozik, amelyben több képesség is megadható.

Coach (edző) egyed

Az edzők alapadatait tárolja, a tapasztalatot években mérve, valamint többértékű *specializations* tulajdonsággal, amely az edző erősségeit jelöli.

TrainingGroup (edzéscsoport) egyed

A csoportokhoz sportág, szint és összetett időbeosztás tartozik (*weekday*, *start_time*, *end_time*).

Location (létesítmény) egyed

A sportlétesítményeket reprezentálja. A *kapacitás* attribútum mellett többértékű *equipment* is tartozik hozzá (pl. kapu, súlyzók, palánk, súlyzórudak).

Membership (tagság)

A sportolók különböző tagsági szintjeit modellezi (standard, premium, elite). A tagsági szinthez havi díj és többértékű *benefits* tulajdonság tartozik.

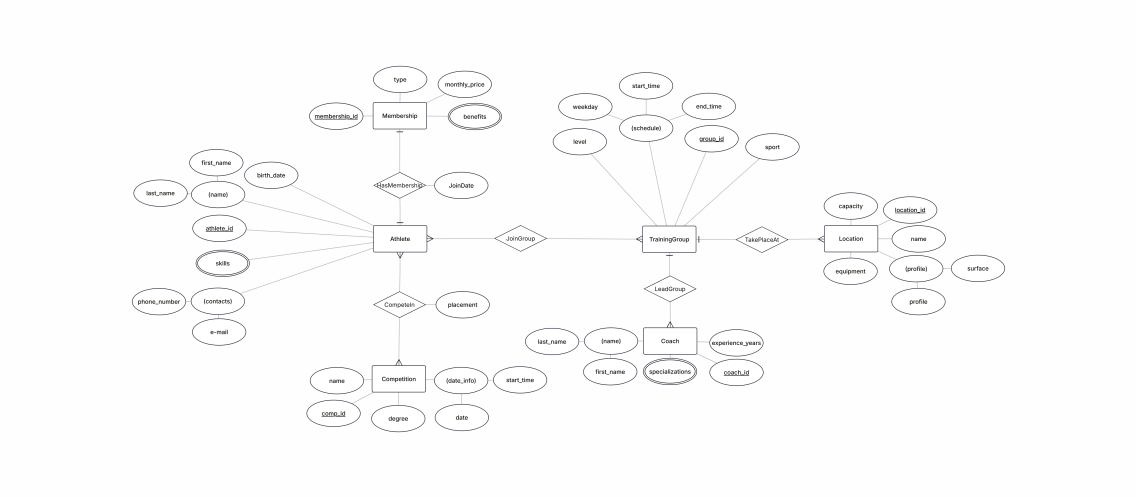
Competition (verseny)

A különböző sportversenyek adatait tárolja, ideértve a dátumot, kezdési időt, helyszínt és kategóriát.

Kapcsolatok

Az egyedek közötti kapcsolatok az alábbiak szerint épülnek fel:

- Athlete–TrainingGroup: N:M (egy sportoló több csoportban is edzhet)
- Coach–TrainingGroup: 1:N (egy edző több csoport vezetője lehet)
- TrainingGroup–Location: 1:N (egy helyszínen több csoport is lehet)
- Athlete–Competition: N:M (több sportoló több versenyen indulhat)
- Athlete–Membership: 1:1 (minden sportolónak egy tagsági rekordja lehet)



1. ábra. ER modell — Sportegyesület

1.2. Az adatbázis konvertálása XDM modellre

Az ER modell XML formátumra történő átültetéséhez XDM (XML Data Model) modellt készítettem. Az XDM célja, hogy az ER diagram elemeit XML elemekre és attribútumokra konvertálja, miközben megtartja az egyedek közötti kapcsolatok szerkezetét.

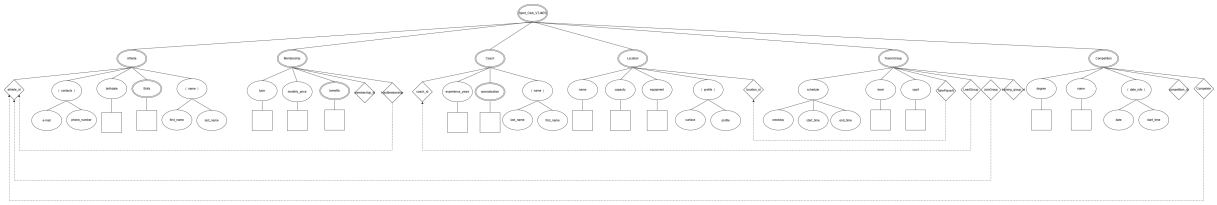
A konvertálás során a következő általános szabályokat alkalmaztam:

- Minden ER egyed külön XML elemként jelenik meg (pl. `<Athlete>`, `<Coach>`).
- Az egyszeres tulajdonságok gyermekelemként kerülnek ábrázolásra.
- Az összetett attribútumok külön al-elemeket kapnak (például a név: `<first_name>`, `<last_name>`).
- A többértékű attribútumok ismétlődő XML elemek formájában jelennek meg.
- Az 1:N kapcsolatokban az idegen kulcs azon az oldalon jelenik meg, ahol az ER modellben a „több” szerepel.
- Az N:M kapcsolatok esetén külön kapcsolat-elemek készülnek (pl. `JoinGroup`, `CompeteIn`).

A sportegyesület XDM modelljének gyökéreleme:

`<SportClub_VTJ4ES>`

A következő ábra mutatja a teljes XDM struktúrát:



2. ábra. XDM modell — Sportegyesület

Az XDM modellben minden kapcsolat jól láthatóan megjelenik, a PK–FK kapcsolatok pedig nem keresztezik egymást, megfelelve a formai előírásoknak. A komplex típusok és többszörös előfordulású elemek is külön jelölést kaptak.

1.3. Az XDM modell alapján XML dokumentum készítése

Az XDM modell alapján elkészítettem a `VTJ4ES_XML.xml` nevű XML dokumentumot, amely a sportegyesület teljes adatbázisát írja le. A dokumentum gyökéreleme: `<Sport_Club_VTJ4ES>`, ezen belül külön elemek reprezentálják az `Athlete`, `Membership`, `Coach`, `Location`, `TrainingGroup` és `Competition` egyedeket. Minden egyedhez attribútumként kerül az azonosító (pl. `athleteID`, `membershipID`, `coachID`), a többi tulajdonság pedig gyermek elemként jelenik meg.

A többértékű tulajdonságokat a minta jegyzőkönyvnek megfelelően többször előforduló elem-részletekkel modelleztem: például a `skills` (sportoló képességei), `benefits` (tagsághoz tartozó jogosultságok) és `equipment` (létesítmény felszerelései) elemek többször szerepelnek ugyanazon szülőelem alatt. Minden ilyen elemnél legalább két példányt hoztam létre.

Az XML 1.0 szabvány szerint készült, `UTF-8` kódolással, és a DOM feldolgozó programok (pl. `VTJ4ESDomRead`) ezt a fájlt használják bemenetként.

Az alábbi kódrészlet a dokumentum elejét és a sportolók, tagságok példányait mutatja:

Listing 1. A Sport_Club_VTJ4ES dokumentum eleje

```
<?xml version="1.0" encoding="UTF-8"?>
<Sport_Club_VTJ4ES>

  <!-- ATHLETE -->
  <Athlete athleteID="1">
    <name>
      <first_name>Daniel</first_name>
      <last_name>Kiss</last_name>
    </name>
    <birth_date>2001-04-10</birth_date>
    <contacts>
      <e_mail>daniel.kiss@example.com</e_mail>
      <phone_number>+36-30-111-1111</phone_number>
    </contacts>
    <skills>speed</skills>
    <skills>endurance</skills>
    <skills>technique</skills>
  </Athlete>

  <Athlete athleteID="2">
    <name>
      <first_name>Bence</first_name>
      <last_name>Nagy</last_name>
    </name>
    <birth_date>1999-09-21</birth_date>
    <contacts>
      <e_mail>bence.nagy@example.com</e_mail>
      <phone_number>+36-30-222-2222</phone_number>
```

```
    </contacts>
    <skills>strength</skills>
    <skills>agility</skills>
    <skills>tactics</skills>
  </Athlete>

  <!-- MEMBERSHIP -->
  <Membership membershipID="1">
    <type>standard</type>
    <monthly_price>12000</monthly_price>
    <benefits>gym_access</benefits>
    <benefits>group_classes</benefits>
    <benefits>locker</benefits>
  </Membership>

  <Membership membershipID="2">
    <type>premium</type>
    <monthly_price>18000</monthly_price>
    <benefits>gym_access</benefits>
    <benefits>sauna</benefits>
    <benefits>massage</benefits>
  </Membership>
```

Hasonló módon kerülnek leírásra az edzők, edzéscsoportok, létesítmények és versenyek. Az alábbi részlet a `Location`, `TrainingGroup` és `Competition` elemek egy-egy példányát mutatja:

Listing 2. Location, TrainingGroup és Competition példányok

```
  <!-- Location -->
  <Location locationID="1">
    <name>Indoor Basketball Court</name>
    <capacity>60</capacity>
    <profile>
      <surface>parketta</surface>
      <profile_text>kosarlabdapalya</profile_text>
    </profile>
    <equipment>basketball_hoop</equipment>
    <equipment>scoreboard</equipment>
    <equipment>benches</equipment>
  </Location>

  <Location locationID="2">
    <name>Outdoor Running Track</name>
    <capacity>120</capacity>
    <profile>
      <surface>synthetic_track</surface>
      <profile_text>futopalya</profile_text>
    </profile>
    <equipment>hurdles</equipment>
    <equipment>starting_blocks</equipment>
    <equipment>timing_system</equipment>
  </Location>
```

```
  <!-- TRAINING GROUPS-->
  <TrainingGroup groupID="1">
    <sport>basketball</sport>
    <level>junior</level>
    <name>Basketball Juniors</name>
    <schedule>
      <weekday>Monday</weekday>
      <start_time>16:00</start_time>
      <end_time>18:00</end_time>
    </schedule>
  </TrainingGroup>

  <TrainingGroup groupID="2">
    <sport>running</sport>
    <level>elite</level>
    <name>Elite Running Team</name>
    <schedule>
      <weekday>Friday</weekday>
      <start_time>07:00</start_time>
      <end_time>09:00</end_time>
    </schedule>
  </TrainingGroup>

  <!-- COMPETITIONS -->
  <Competition compID="1">
    <name>Kiss Daniel Sprint Meet</name>
    <degree>regional</degree>
    <date_info>
      <date>2025-05-10</date>
      <start_time>10:00</start_time>
    </date_info>
  </Competition>

  <Competition compID="2">
    <name>Nagy Bence Basketball Cup</name>
    <degree>national</degree>
    <date_info>
      <date>2025-06-20</date>
      <start_time>18:00</start_time>
    </date_info>
  </Competition>

</Sport_Club_VTJ4ES>
```

A teljes XML dokumentum a projektben a `VTJ4ES_XML.xml` fájlban található meg.

1.4. Az XML dokumentum alapján XMLSchema készítése

Az XML dokumentumhoz egy külön XML Schema állomány (`VTJ4ES_XMLSchema.xsd`) készült, amely leírja, hogy a `Sport_Club_VTJ4ES` gyökérelem alatt milyen elemek, attri-

bútumok és típusok fordulhatnak elő, illetve milyen megszorításoknak kell megfelelniük. A séma a minta jegyzőkönyv elvárásainak megfelelően tartalmaz:

- saját egyszerű típusokat (pl. azonosítókra és pozitív egész számokra),
- komplex típusokat az egyes egyedekhez (`AthleteType`, `MembershipType`, stb.),
- `xs:key` és `xs:unique` kulcsdefiníciókat az azonosítókra,
- szükség esetén `xs:keyref` hivatkozásokat kapcsolatok modellezésére.

A gyökérelem, illetve a főbb típusok definíciójának részlete:

Listing 3. A VTJ4ES_XMLSchema.xsd séma eleje

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- simple types for IDs and positive integers -->
  <xs:simpleType name="idType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="positiveIntType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- complex type for person name -->
  <xs:complexType name="NameType">
    <xs:sequence>
      <xs:element name="first_name" type="xs:string"/>
      <xs:element name="last_name" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <!-- complex type for contacts -->
  <xs:complexType name="ContactsType">
    <xs:sequence>
      <xs:element name="e_mail" type="xs:string"/>
      <xs:element name="phone_number" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

A sportolókat leíró komplex típus és a többértékű `skills` elemek kezelése:

Listing 4. AthleteType komplex típus

```
  <!-- complex type for Athlete -->
  <xs:complexType name="AthleteType">
    <xs:sequence>
      <xs:element name="name" type="NameType"/>
```

```
    <xs:element name="birth_date" type="xs:date"/>
    <xs:element name="contacts" type="ContactsType"/>
    <xs:element name="skills" type="xs:string" minOccurs="1"
       maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="athleteID" type="idType" use="required"/>
</xs:complexType>
```

A tagság, létesítmény, edzéscsoport és verseny típusai hasonló struktúrát követnek. Például a tagsági rekord és az edzéscsoport:

Listing 5. MembershipType és TrainingGroupType részlete

```
<!-- complex type for Membership -->
<xs:complexType name="MembershipType">
  <xs:sequence>
    <xs:element name="type" type="xs:string"/>
    <xs:element name="monthly_price" type="positiveIntType"/>
    <xs:element name="benefits" type="xs:string" minOccurs="1"
       maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="membershipID" type="idType" use="required"/
     >
</xs:complexType>

<!-- complex type for TrainingGroup -->
<xs:complexType name="TrainingGroupType">
  <xs:sequence>
    <xs:element name="sport" type="xs:string"/>
    <xs:element name="level" type="xs:string"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="schedule">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="weekday" type="xs:string"/>
          <xs:element name="start_time" type="xs:string"/>
          <xs:element name="end_time" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="groupID" type="idType" use="required"/>
</xs:complexType>
```

Végül a gyökérelem definiálása és az egyedi kulcsok beállítása:

Listing 6. Gyökérelem és kulcsdefiníciók

```
<!-- root element -->
<xs:element name="Sport_Club_VTJ4ES">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Athlete" type="AthleteType" minOccurs="0"
         maxOccurs="unbounded"/>
```

```
        <xs:element name="Membership" type="MembershipType"
           minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Coach" type="CoachType" minOccurs="0"
           maxOccurs="unbounded"/>
        <xs:element name="Location" type="LocationType" minOccurs="
           0" maxOccurs="unbounded"/>
        <xs:element name="TrainingGroup" type="TrainingGroupType"
           minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Competition" type="CompetitionType"
           minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <!-- keys for primary identifiers -->
    <xs:key name="AthleteKey">
      <xs:selector xpath="Athlete"/>
      <xs:field xpath="@athleteID"/>
    </xs:key>

    <xs:key name="MembershipKey">
      <xs:selector xpath="Membership"/>
      <xs:field xpath="@membershipID"/>
    </xs:key>

    <xs:key name="CoachKey">
      <xs:selector xpath="Coach"/>
      <xs:field xpath="@coachID"/>
    </xs:key>

    <xs:key name="LocationKey">
      <xs:selector xpath="Location"/>
      <xs:field xpath="@locationID"/>
    </xs:key>

    <xs:key name="GroupKey">
      <xs:selector xpath="TrainingGroup"/>
      <xs:field xpath="@groupID"/>
    </xs:key>

    <xs:key name="CompetitionKey">
      <xs:selector xpath="Competition"/>
      <xs:field xpath="@compID"/>
    </xs:key>

  </xs:element>

</xs:schema>
```

A séma tehát biztosítja, hogy az XML dokumentum minden azonosítója egyedi legyen, a többértékű tulajdonságok megfelelően ismétlődő elemekként jelenjenek meg, és a `Sport_Club_VTJ4ES` struktúrája megfeleljen az 1. feladatban tervezett ER és XDM mo-

dellnek. A teljes sémafájl a projektben a `VTJ4ES_XMLSchema.xsd` néven található meg.

2. II. feladat – DOM feldolgozás

A projekt második részében a `VTJ4ES_XML.xml` dokumentum feldolgozása történik Java nyelven a DOM (Document Object Model) API használatával. A DOM egy faalapú reprezentációt hoz létre az XML állományról, amelyben az elemek, attribútumok és szöveges csomópontok külön-külön, egymáshoz hierarchikusan kapcsolódó objektumokként jelennek meg. Ennek köszönhetően a teljes dokumentum struktúrája tetszőlegesen bejárható, módosítható, majd újra kiírható.

A feladat célja az volt, hogy a sportegyesület XML alapú adatbázisán a gyakorlatban is bemutassam a DOM-alapú feldolgozás főbb műveleteit:

- **beolvasás**: a dokumentum betöltése és normalizálása DOM fába,
- **lekérdezés**: különböző szempontok szerinti adatkiválasztás,
- **módosítás**: meglévő elemek és értékek átalakítása,
- **kiírás**: a módosított DOM fa visszaírása fájlba.

A feldolgozás négy külön osztályban valósul meg, amelyek egyértelműen szétválasztják a funkciókat:

- `VTJ4ESDomRead.java` – XML beolvasása és strukturált kiírása,
- `VTJ4ESDomQuery.java` – több, XML-ből készített lekérdezés,
- `VTJ4ESDomModify.java` – módosítások végrehajtása a DOM fán,
- `VTJ4ESDomWrite.java` – DOM tartalom kiírása fájlba.

Az alfejezetekben először röviden szövegesen összefoglalom az egyes programok működését, majd a legfontosabb kódrészletek kerülnek bemutatásra.

2.1. 2.1 Adatolvasás – VTJ4ESDomRead.java

A VTJ4ESDomRead osztály feladata a teljes XML dokumentum beolvasása, normalizálása, majd az egyes egyedtípusok (Athlete, Membership, Coach, Location, TrainingGroup, Competition) kiírása jól áttekinthető, blokkszerű formában a konzolra. A program célja elsősorban az, hogy visszaellenőrizhető legyen: a VTJ4ES_XML.xml állomány valóban azt a szerkezetet és tartalmat tartalmazza, amit az 1. feladatban megtervezett modell előír.

A main metódusban létrejön a DocumentBuilderFactory és a DocumentBuilder, majd az XML fájl egy Document objektumba kerül betöltésre. A setIgnoringElementContentWhitespa hívás biztosítja, hogy a felesleges whitespace csomópontok ne zavarják a feldolgozást, a normalize() pedig egységesíti a DOM-fa belső reprezentációját.

A program ezután kiírja az XML deklarációt és a gyökérelem nyitó tagjét, majd egymás után meghívja a readAthletes, readMemberships, readCoaches, readLocations, readTrainingGroups és readCompetitions metódusokat. Ezek mindegyike egy adott egyedtípushoz tartozó NodeList-en iterál, kiolvassa az attribútumokat és al-elemeket, majd az printElement segédfüggvény segítségével XML-szerű formában visszaírja a konzolra.

Az alábbi kódrészlet az osztály elejét és a main metódust mutatja:

Listing 7. VTJ4ESDomRead – main metodus

```
package VTJ4ES.domparse.hu;

import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.w3c.dom.*;
import java.io.*;

public class VTJ4ESDomRead {

    // Main method
    public static void main(String[] args) {
        try {
            // Create DOM parser
            DocumentBuilderFactory factory = DocumentBuilderFactory
               .newInstance();
            factory.setIgnoringElementContentWhitespace(true);
            DocumentBuilder builder = factory.newDocumentBuilder();

            // Load XML file (change path if needed)
            Document document = builder.parse(new File("./
               VTJ4ES_XML.xml"));

            // Normalize document
            document.getDocumentElement().normalize();

            // Print XML declaration and root start tag with XSD
               reference
            System.out.println("<?xml␣version=\"1.0\"␣encoding=\"
               UTF-8\"?>\n");
            System.out.println("<Sport_Club_VTJ4ES␣xmlns:xsi=\"http
               ://www.w3.org/2001/XMLSchema-instance\"␣xsi:
```

```
                noNamespaceSchemaLocation=\"./VTJ4ES_XMLSchema.xsd
                \">\n");

            // Read and print each entity type
            readAthletes(document);
            readMemberships(document);
            readCoaches(document);
            readLocations(document);
            readTrainingGroups(document);
            readCompetitions(document);

            // Closing root tag
            System.out.println("\n</Sport_Club_VTJ4ES>");

        } catch (ParserConfigurationException | IOException |
            SAXException e) {
            e.printStackTrace();
        }
    }
```

A sportolók kiolvasását végző `readAthletes` metódus bemutatja, hogyan érhetők el az attribútumok (pl. `athleteID`) és az összetett, illetve többértékű mezők (például a `name` és a `skills`):

Listing 8. VTJ4ESDomRead – readAthletes

```
    // Read Athlete nodes
    private static void readAthletes(Document document) {
        NodeList athleteList = document.getElementsByTagName("
            Athlete");
        for (int i = 0; i < athleteList.getLength(); i++) {
            Node node = athleteList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element e = (Element) node;

                String athleteId = e.getAttribute("athleteID");

                Element nameElem = (Element) e.getElementsByTagName
                    ("name").item(0);
                String firstName = nameElem.getElementsByTagName("
                    first_name").item(0).getTextContent();
                String lastName = nameElem.getElementsByTagName("
                    last_name").item(0).getTextContent();

                String birthDate = e.getElementsByTagName("
                    birth_date").item(0).getTextContent();

                Element contactsElem = (Element) e.
                    getElementsByTagName("contacts").item(0);
                String email = contactsElem.getElementsByTagName("
                    e_mail").item(0).getTextContent();
                String phone = contactsElem.getElementsByTagName("
                    phone_number").item(0).getTextContent();
```

```
                System.out.println("␣␣␣␣<Athlete␣athleteID=\"" +
                    athleteId + "\">");
                System.out.println("␣␣␣␣␣␣␣␣<name>");
                printElement("first_name", firstName);
                printElement("last_name", lastName);
                System.out.println("␣␣␣␣␣␣␣␣</name>");
                printElement("birth_date", birthDate);
                System.out.println("␣␣␣␣␣␣␣␣<contacts>");
                printElement("e_mail", email);
                printElement("phone_number", phone);
                System.out.println("␣␣␣␣␣␣␣␣</contacts>");

                // Multivalued 'skills'
                NodeList skills = e.getElementsByTagName("skills");
                for (int j = 0; j < skills.getLength(); j++) {
                    String skill = skills.item(j).getTextContent();
                    printElement("skills", skill);
                }

                System.out.println("␣␣␣␣</Athlete>");
            }
        }
    }

    // Helper method to print elements
    private static void printElement(String tagName, String content
        ) {
        System.out.println("␣␣␣␣␣␣␣␣<" + tagName + ">" + content +
            "</" + tagName + ">");
    }
}
```

A többi `readX` metódus ugyanazt a mintát követi: a megfelelő tag-ek `NodeList`-jén végighalad, az elemeket `Element`-re castolja, kiolvassa a gyermek elemek szövegét, majd a konzolra formázott XML-ként írja ki. Ez jól demonstrálja a DOM-fa bejárását és az adatok strukturált feldolgozását.

2.2. 2.2 Adatlekérdezés – VTJ4ESDomQuery.java

A VTJ4ESDomQuery osztály különböző, egymástól jól elkülöníthető lekérdezéseket valósít meg a DOM fa bejárásával. A feladat kiírásának megfelelően XPath használata nélkül, tisztán NodeList-eken és Element-objektumokon végzett ciklusokkal történik az adatok szűrése. A lekérdezések eredményeit egy StringBuilder gyűjti össze, XML-szerű formában, amelyet a program a futás végén a konzolra ír ki.

A main metódus felépítése hasonló a VTJ4ESDomRead-hez: létrejön a DOM parser, beolvasásra kerül a VTJ4ES_XML.xml, majd a normalize() hívás után elkészül az üres StringBuilder, amelybe a lekérdezések blokkjai kerülnek.

Az első lekérdezés a skills között a "speed" értéket keresi, és csak azokat a sportolókat írja ki, akiknél ez a képesség szerepel. A második lekérdezés a TrainingGroup elemek közül a running sportágú csoportokat gyűjti ki, a hozzájuk tartozó időbeosztással együtt. A további lekérdezések hasonlóan működnek: egyik például a national szintű versenyeket válogatja ki, másik pedig a 2000 után született sportolókat listázza.

Az alábbi részlet a dokumentum beolvasását és az első két lekérdezést mutatja:

Listing 9. VTJ4ESDomQuery – main eleje

```
package VTJ4ES.domparse.hu;

import java.io.File;
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class VTJ4ESDomQuery {

    // Main method
    public static void main(String[] argv) throws SAXException,
       IOException, ParserConfigurationException {

        File xmlFile = new File("./VTJ4ES_XML.xml");

        DocumentBuilderFactory factory = DocumentBuilderFactory.
           newInstance();
        factory.setIgnoringElementContentWhitespace(true);
        DocumentBuilder dBuilder = factory.newDocumentBuilder();

        Document doc = dBuilder.parse(xmlFile);
        doc.getDocumentElement().normalize();

        StringBuilder outputBuilder = new StringBuilder();
```

Listing 10. VTJ4ESDomQuery – Query 1 és 2

```
        // Query 1: Athletes with skill = "speed"
        NodeList athleteList = doc.getElementsByTagName("Athlete");
        outputBuilder.append("<AthletesWithSpeed>\n");
        for (int i = 0; i < athleteList.getLength(); i++) {
            Node node = athleteList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
```

```
            Element athlete = (Element) node;

            NodeList skills = athlete.getElementsByTagName("
               skills");
            boolean hasSpeed = false;
            for (int j = 0; j < skills.getLength(); j++) {
                String skillValue = skills.item(j).
                   getTextContent();
                if ("speed".equalsIgnoreCase(skillValue)) {
                    hasSpeed = true;
                    break;
                }
            }

            if (hasSpeed) {
                String athleteId = athlete.getAttribute("
                   athleteID");
                Element nameElem = (Element) athlete.
                   getElementsByTagName("name").item(0);
                String firstName = nameElem.
                   getElementsByTagName("first_name").item(0).
                   getTextContent();
                String lastName = nameElem.getElementsByTagName
                   ("last_name").item(0).getTextContent();
                String birthDate = athlete.getElementsByTagName
                   ("birth_date").item(0).getTextContent();

                outputBuilder.append(String.format("␣␣<Athlete␣
                   athleteID=\"%s\">\n", athleteId));
                outputBuilder.append(String.format("␣␣␣␣<
                   first_name>%s</first_name>\n", firstName));
                outputBuilder.append(String.format("␣␣␣␣<
                   last_name>%s</last_name>\n", lastName));
                outputBuilder.append(String.format("␣␣␣␣<
                   birth_date>%s</birth_date>\n", birthDate));
                outputBuilder.append("␣␣␣␣<skills>speed</skills
                   >\n");
                outputBuilder.append("␣␣</Athlete>\n");
            }
        }
    }
    outputBuilder.append("</AthletesWithSpeed>\n");

    // Query 2: Training groups with sport = "running"
    NodeList groupList = doc.getElementsByTagName("
       TrainingGroup");
    outputBuilder.append("\n<RunningGroups>\n");
    for (int i = 0; i < groupList.getLength(); i++) {
        Node node = groupList.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            Element group = (Element) node;
```

```
                String sport = group.getElementsByTagName("sport").
                    item(0).getTextContent();

                if ("running".equalsIgnoreCase(sport)) {
                    String groupId = group.getAttribute("groupID");
                    String level = group.getElementsByTagName("
                        level").item(0).getTextContent();
                    String name = group.getElementsByTagName("name"
                        ).item(0).getTextContent();

                    Element schedule = (Element) group.
                        getElementsByTagName("schedule").item(0);
                    String weekday = schedule.getElementsByTagName(
                        "weekday").item(0).getTextContent();
                    String startTime = schedule.
                        getElementsByTagName("start_time").item(0).
                        getTextContent();
                    String endTime = schedule.getElementsByTagName(
                        "end_time").item(0).getTextContent();

                    outputBuilder.append(String.format("␣␣<
                        TrainingGroup␣groupID=\"%s\">\n", groupId));
                    outputBuilder.append(String.format("␣␣␣␣<name>%
                        s</name>\n", name));
                    outputBuilder.append(String.format("␣␣␣␣<sport
                        >%s</sport>\n", sport));
                    outputBuilder.append(String.format("␣␣␣␣<level
                        >%s</level>\n", level));
                    outputBuilder.append("␣␣␣␣<schedule>\n");
                    outputBuilder.append(String.format("␣␣␣␣␣␣<
                        weekday>%s</weekday>\n", weekday));
                    outputBuilder.append(String.format("␣␣␣␣␣␣<
                        start_time>%s</start_time>\n", startTime));
                    outputBuilder.append(String.format("␣␣␣␣␣␣<
                        end_time>%s</end_time>\n", endTime));
                    outputBuilder.append("␣␣␣␣</schedule>\n");
                    outputBuilder.append("␣␣</TrainingGroup>\n");
                }
            }
        }
        outputBuilder.append("</RunningGroups>\n");
```

Minden lekérdezés ugyanarra az alapmintára épül:

- a megfelelő elemnévvel (`Athlete`, `TrainingGroup`, `Competition` stb.) lekérdezünk egy `NodeList`-et,
- végigiterálunk rajta, és minden `ELEMENT_NODE`-ot `Element` típusra castolunk,
- az al-elemekből (`getElementsByTagName`) kiolvassuk a szükséges adatokat,
- feltételek alapján (pl. skill = "speed", sport = "running", degree = "national", születési év $>$ 2000) eldöntjük, hogy az adott példány bekerüljön-e a kimenetbe,

- az eredményt XML szerűen hozzáfűzzük a `StringBuilder`-hez.

Ez a megoldás jól tükrözi a DOM modell manuális bejárásának gyakorlatát és a saját logika szerinti szűrések felépítését.

2.3. 2.3 Adatmódosítás – VTJ4ESDomModify.java

A VTJ4ESDomModify osztály már nem csak olvassa, hanem aktívan módosítja is a DOM fát. A program bemenetként ismét a VTJ4ES_XML.xml fájlt használja, azonban a csomópontok értékeit és egyes attribútumokat megváltoztatja, majd az így módosított dokumentumot formázott XML-ként a konzolra kiírja.

A main metódusban a DOM beolvasása és normalizálása után egymás után meghívódik három segédfüggvény:

- modifyAthletes – az athleteID attribútumok "ATH_" prefixet kapnak, így könnyen megkülönböztethetők,
- modifyMemberships – a monthly_price értékek egységesen megnövekednek (pl. +1000 Ft),
- modifyLocations – a capacity minimális értéket kap, illetve a profile_text módosul.

Mindezt tisztán DOM műveletekkel valósítja meg: getElementsByTagName, getAttribute, setAttribute, getTextContent, setTextContent. A módosítások után egy Transformer segítségével a Document objektum tartalma XML formátumban a konzolra kerül.

Az alábbi kódrészlet a belépési pontot mutatja:

Listing 11. VTJ4ESDomModify – main

```
package VTJ4ES.domparse.hu;

import javax.xml.parsers.*;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.*;

import java.io.File;

public class VTJ4ESDomModify {

    // Main method
    public static void main(String[] argv) {
        try {
            // Input XML file (place VTJ4ES_XML.xml in the project
               root)
            File inputFile = new File(".//VTJ4ES_XML.xml");

            // Build DOM document
            DocumentBuilderFactory docFactory =
               DocumentBuilderFactory.newInstance();
            docFactory.setIgnoringElementContentWhitespace(true);
            DocumentBuilder docBuilder = docFactory.
               newDocumentBuilder();
```

```
            Document doc = docBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();

            // Perform modifications
            modifyAthletes(doc);
            modifyMemberships(doc);
            modifyLocations(doc);

            // Print modified DOM to console as formatted XML
            TransformerFactory transformerFactory =
               TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.
               newTransformer();

            transformer.setOutputProperty(OutputKeys.INDENT, "yes")
               ;
            transformer.setOutputProperty("{http://xml.apache.org/
               xslt}indent-amount", "2");
            transformer.setOutputProperty(OutputKeys.ENCODING, "UTF
               -8");

            DOMSource source = new DOMSource(doc);
            StreamResult consoleResult = new StreamResult(System.
               out);
            transformer.transform(source, consoleResult);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

A fő módosító metódusok:

Listing 12. VTJ4ESDomModify – modifyAthletes / modifyMemberships

```
    // Modify Athlete elements: prefix athleteID with "ATH_"
    private static void modifyAthletes(Document doc) {
        NodeList athleteList = doc.getElementsByTagName("Athlete");

        for (int i = 0; i < athleteList.getLength(); i++) {
            Node node = athleteList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element athlete = (Element) node;
                String id = athlete.getAttribute("athleteID");

                // Avoid double prefixing if run multiple times
                if (id != null && !id.startsWith("ATH_")) {
                    athlete.setAttribute("athleteID", "ATH_" + id);
                }
            }
        }
    }
```

```
    // Modify Membership elements: increase monthly_price by 1000
    private static void modifyMemberships(Document doc) {
        NodeList membershipList = doc.getElementsByTagName("
            Membership");

        for (int i = 0; i < membershipList.getLength(); i++) {
            Node node = membershipList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element membership = (Element) node;

                Node priceNode = membership.getElementsByTagName("
                    monthly_price").item(0);
                if (priceNode != null && priceNode.getNodeType() ==
                    Node.ELEMENT_NODE) {
                    String priceText = priceNode.getTextContent().
                        trim();
                    try {
                        int price = Integer.parseInt(priceText);
                        price += 1000; // increase by 1000
                        priceNode.setTextContent(String.valueOf(
                            price));
                    } catch (NumberFormatException e) {
                        // If the value is not a valid integer,
                            leave it unchanged
                    }
                }
            }
        }
    }
```

A `modifyLocations` metódus hasonló logikával működik: egy küszöbértékhez hasonlítja a `capacity` mezőt, és ha az ez alatti, akkor módosítja, illetve a `profile_text` végére egy `" (updated)"` kiegészítést tesz. Ez a rész jól szemlélteti, hogyan lehet a DOM fán tömeges, szabály-alapú módosításokat végrehajtani.

2.4. 2.4 Adatírás – VTJ4ESDomWrite.java

A VTJ4ESDomWrite osztály a DOM-ból történő visszaírás folyamatát mutatja be. A program ismét beolvassa a VTJ4ES_XML.xml állományt DOM-faként, majd egyrészt a konzolra, másrészt egy új fájlba is kiírja ugyanennek a dokumentumnak a tartalmát. A megoldás így egyszerre demonstrálja a DOM alapú olvasást, valamint a Transformer API alapvető használatát.

A main metódusban először létrejön a DOM parser, majd a dokumentum beolvasása és normalizálása után egy Transformer objektum példányosodik. Ennek kimeneti tulajdonságai közül a legfontosabb az INDENT, amely gondoskodik a szépen formázott (behúzott) XML kiírásról, illetve az indent-amount, amely a behúzás mértékét bájtokban adja meg.

Első lépésben a DOMSource – StreamResult(System.out) páros segítségével a teljes dokumentum a konzolra íródik. Ez különösen hasznos a módosítások ellenőrzéséhez. Ezt követően a writeDocumentToFile segédfüggvény hívásával ugyanaz a Document egy új fájlba (VTJ4ES_XML1.xml) kerül elmentésre, így az eredeti bemeneti fájl érintetlen marad.

Az alábbi kódrészlet a main metódust mutatja:

Listing 13. VTJ4ESDomWrite – main

```
package VTJ4ES.domparse.hu;

import org.w3c.dom.Document;
import org.xml.sax.SAXException;

import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.IOException;

public class VTJ4ESDomWrite {

    // Main method
    public static void main(String[] args) {
        try {
            // Input XML file (place VTJ4ESXML.xml in the project
               root)
            File inputFile = new File("./VTJ4ES_XML.xml");

            // Create DOM parser
            DocumentBuilderFactory dbFactory =
               DocumentBuilderFactory.newInstance();
            dbFactory.setIgnoringElementContentWhitespace(true);
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder
               ();

            // Parse XML into DOM
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();

            // Print XML declaration
```

```
            System.out.println("<?xml␣version=\"1.0\"␣encoding=\"
               UTF-8\"?>");

            // Print DOM content to console
            TransformerFactory transformerFactory =
               TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.
               newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes")
               ;
            transformer.setOutputProperty("{http://xml.apache.org/
               xslt}indent-amount", "2");

            DOMSource source = new DOMSource(doc);
            StreamResult consoleResult = new StreamResult(System.
               out);
            transformer.transform(source, consoleResult);

            // Write DOM content to a new XML file
            writeDocumentToFile(doc, "./VTJ4ES_XML1.xml");

            System.out.println("\nThe␣content␣has␣been␣written␣to␣
               VTJ4ESXML1.xml␣successfully.");

        } catch (SAXException | IOException |
           ParserConfigurationException | TransformerException e) {
            e.printStackTrace();
        }
    }

    // Write DOM document to output file
    private static void writeDocumentToFile(Document doc, String
       filename) throws TransformerException {
        TransformerFactory transformerFactory = TransformerFactory.
           newInstance();
        Transformer transformer = transformerFactory.newTransformer
           ();

        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.setOutputProperty("{http://xml.apache.org/xslt}
           indent-amount", "2");
        transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8")
           ;

        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File(filename));
        transformer.transform(source, result);
    }
}
```

A `writeDocumentToFile` függvényben ugyanaz a transzformáció hajtódik végre, csak a kimenet ezúttal egy fájl, nem pedig a konzol. A konfiguráció szándékosan egyszerű, jól

követhető, és illeszkedik a minta jegyzőkönyvben elvárt DOMWrite feladathoz: beolvasás, opcionális módosítás, formázott kiírás, majd a dokumentum lementése.